A Hybrid Topology Architecture for P2P Systems

Aameek Singh
Georgia Institute of Technology
aameek@cc.gatech.edu

Ling Liu
Georgia Institute of Technology
lingliu@cc.gatech.edu

Abstract

A core area of P2P systems research is the topology of the overlay network. It has ranged from random unstructured networks like Gnutella [8] to Super-Peer [9] architectures to the recent trend of structured overlays based on Distributed Hash Tables (DHTs) [4, 12, 11]. While the unstructured networks have excessive lookup costs and un-guaranteed lookups, the structured systems offer no anonymity and delegate control over data items to unrelated peers. In this paper, we present an in-the-middle hybrid architecture which uses a mix of both topologies to create a decentralized P2P infrastructure. The system provides scalable and guaranteed lookups in addition to mutual anonymity and also allows hosting content with the content-owner. We validate our architecture through a thorough analytical and empirical performance analysis of the system.

1 Introduction

In recent years, Peer-to-Peer (P2P) systems have received a great deal of attention from both research and industrial communities. The paradigm of allowing users to provide and avail services directly without any intermediaries, has seen successful applications in file sharing [8, 9, 2], distributed computing [13, 7], data dissemination [1, 6] and many other areas [14, 10]. The important characteristics of P2P systems are its network topology and routing protocols. While Gnutella like systems follow an unstructured random overlay without any system policy to enforce topology, the new generation P2P systems based on DHTs like Chord [4], Pastry [12], CAN [11] enforce certain topologies to provide guarantees of lookup and bounds on lookup costs.

However, both kinds of topologies have their advantages and disadvantages. We compare the two on three important P2P characteristics:

1. Lookups: The unstructured Gnutella like systems follow a network flooding mechanism in which a query is either sent to every peer in the network, which is extremely expensive or is prematurely terminated after a certain number of hops leading to un-guaranteed lookups. On the other hand, the structured systems map every data item to some peer and store that mapping in distributed hash tables (routing tables). Their routing protocols are proven to be scalable in both number of hops and state information maintained at peers.

2. Anonymity: The unstructured systems follow a message forwarding protocol and keep local knowledge at peers (information about neighbours only). This provides an important feature of *anonymity*, the word that has almost become synonymous with P2P systems. This anonymity is provided by the inability of peers to identify where any message originates or terminates, since the *local knowledge* property ensures that a peer only knows the neighbor, it received the message from, without any possibility of knowing if that neighbour initiated it or if it is just forwarding a message it received further from one of its neighbors. Similary for termination of messages. A peer never knows whether the message terminated at its neighbor or was forwarded further into the network.
It is easy to see that structured DHT based systems do not provide anonymity. Since they map data items directly to peers and store this information in routing tables, it is easy to identify the destination of a message by looking at the data item it is requesting.

3. Content Hosting: In unstructured systems, the content is hosted by the peer who owns the content. The owner replies for the queries for that content and controls all access to it. On the other hand, in DHT based systems, the access to content is controlled by a peer where that data item hashes onto using the DHT based policy, which is very unlikely to be the owner peer.

In this paper, we provide an architecture of a decentralized P2P system, which uses a mix of both structured and unstructured topologies, ensuring the *nice* properties of both. It provides guaranteed and scalable lookups, while also ensuring mutual anonymity, protecting the privacy of both service requester and service providers. In addition, it follows similar content hosting semantics as unstructured systems, allowing complete control of data with its owner. The paper is organized as follows. We present the design of the system in Section-2. We analyze the system performance in Section-3 and conclude in Section-4.

2 System Design

We first identify the features of structured and unstructured topologies that allow for the aforementioned properties. It is easy to see how unstructured systems provide anonymity. Whenever Peer-A receives a message from Peer-B, the only information about its origin is that it came from the *group* of peers, Peer-B is connected to. And the information with Peer-B about its destination is the group of peers, Peer-A is connected to. Thus, the responsibility of origin/destination is cloaked by spreading it to a group. This presents an interesting insight. If instead of mapping data items to peers, as done in current DHT based systems, we map data items to a group of peers, we *can* provide mutual anonymity. Also interestingly, we only need to have these groups at the query end-points, i.e. only the origin and destination needs to be cloaked. Thus, for routing within these groups, we can use the scalable routing mechanisms of the DHT based systems. In addition, mapping items to groups also allows for flexible content hosting. Now, an owner can keep a data item as long as the data item hashes onto the group it is in. The way in which we achieve this, is described in detail later.

The above mentioned insights serve as the main motivation for our system. We provide mutual anonymity by adding-on unstructured Gnutella-like topologies on top of a DHT-based P2P network and mapping items onto these groups. We call such topologies **clouds**, for the cloaking effect they have to shield the service requester and service provider. In addition, we use normal DHT-based routing to link a service requester's cloud to a service provider's cloud to ensure good lookup performance. Now, every peer will initiate a query within a cloud (provides anonymity); the query will then go onto the DHT ring to reach the service provider's cloud (good lookup performance); and is transmitted to the service provider in Gnutella-like fashion (anonymity). The reply is similarly traced back. Such a design associates with itself a host of challenging issues, including *(a)* how to maintain routing properties in spite of two different topologies - the unstructured clouds and the underlying DHT overlay and *(b)* how to ensure scalable and guaranteed lookups with performance comparable to Chord [4] like pure DHT systems.

2.1 Clouds: Creation and Routing

In our system, all peers, in addition to being a part of the DHT ring, form small unstructured clouds by connecting to other peers in a Gnutella like fashion. For example, Figure-1 shows two clouds with nodes being part of both the cloud and the DHT ring (blue nodes form one cloud and white nodes another).

A cloud is a small unstructured Gnutella-like network. Since now a *key* is mapped to a cloud using normal DHT operations, it is essential to represent the clouds on the DHT ring i.e. to find an entry point into the cloud. This is accomplished using the concept of rendezvous nodes, similar to the work done in P2P multicast [1, 6]. Each cloud has a name and using some hash function, it is hashed onto the DHT ring. The node responsible for that region (according to the normal DHT policy) is found and it acts as an entry point into the cloud. This node is called the *rendezvous node* and is required to be a member of that cloud. Because of the consistent hashing properties of the DHT based systems [3], the load on a node due to its rendezvous properties will be approximately equally distributed amongst all the nodes. Also, in case some rendezvous node leaves, a new one is found by virtue of the dynamics handling of the DHT protocols and it simply replaces the old one in the cloud. Note that given a cloud name, it is always possible to find its rendezvous nodes, by just doing a DHT lookup for its hashed name.

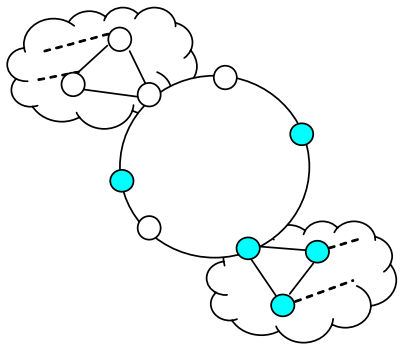

Figure 1: Add-on Clouds over DHTs

To create a cloud, the desired name for the cloud is hashed onto the ring using multiple hash functions and thus multiple rendezvous nodes are found. These nodes connect to each other to form a small Gnutella network. The number of hash functions used depends upon the desired initial membership of a cloud and can be set as a system parameter[1]. Any node wishing to join a cloud, uses any one of the hash functions to get to a rendezvous node which then bootstraps it into the cloud, similar to the bootstrapping process of Gnutella. To allow peers to get a list of clouds currently active in the network, each peer caches the cloud names it sees queries/replies from and this list is shared with an incoming peer.

An important question still remains. How can we link the service provided by *any* member of the cloud to that cloud? It is essential since we need a mechanism ensuring that a query for a service reaches the cloud of the service provider. This can be a complicated task de-

[1]This number effects the amount of initial anonymity of the cloud. For example, if there was only one rendezvous node, then clearly at the time of cloud creation, no immediate anonymity can be provided.

pending upon the kind of services being supported by the system. We have classified the types of services into three categories:
1. Semantic Groups: This is a kind of service in which services being offered by peers in a cloud are semantically linked to each other. For example, for a file sharing application, the peers belonging to a same cloud could be sharing music from a single artist. The artist's name is used as the cloud name and queries for its songs are tagged with the cloud name. The DHT lookup will lead to the rendezvous node for that cloud and the query is forwarded to it.
2. Services with Discovery of Service Mechanisms: While it may be possible to link a query to a cloud semantically for many cases, there may be cases when it is not possible, for example, only the song is known and not its artist. For such a group of services, there might exist a discovery of service mechanism, which links the query item to a cloud, e.g. a central directory service. This category is actually a generalization of the semantic groups category, in which the discovery was due to the semantic nature of the services being offered.
3. Dynamic Services: This is a class of services without a service discovery mechanism or where it is prohibitory to use a centralized mechanism. For example, a co-operative decentralized web crawling application like [14], where peers dynamically decide which web site to crawl accordingly to a DHT based system policy. In such a case, it is not feasible to have a centralized directory service for the prohibitory performance costs. Now, for a given URL there is no way to determine the name of the cloud responsible for crawling that URL, which is important to prevent repeated crawling of the URLs. For such services, we use *R-Rings*.
Notice that for the first two categories, it is the responsibility of the peer to join an appropriate cloud. For the third category, the peer can join any cloud.

2.1.1 Mapping Services to Clouds: R-Rings

Let us take a look at mechanisms that can be used to support dynamic services. One of the ways is by just assigning the data item to the cloud of the peer, it hashes onto using the DHT policies. For the web crawling example, if the URL hashes onto some Peer-A, then the cloud, of which Peer-A is a member of, will be responsible for crawling that URL. Clearly this is not an efficient policy, because of the fact that peers can be very dynamic. Thus if Peer-A leaves the system, and now the URL hashes onto Peer-B, then a new cloud might become responsible. To make this work, we will have to maintain lots of state information to remember the cloud name it hashed onto previously. This will quickly become unscalable.

This problem occurs since the routing of query items is based on peers, which tend to be very dynamic in nature. On the other hand, a cloud is static and it persists even when existing members leave or new ones join. Therefore, any mechanism in which the routing occurs based on cloud names, will be successful to handle this issue. This leads to the idea of *Rendezvous Rings*.

An R-Ring is a special DHT ring, consisting of one rendezvous node from each cloud. The idea is to create a smaller ring comprising only of representatives of the clouds (rendezvous node) and each representative takes up a position on the ring dictated not by its identity but by the name of the cloud. Thus, while the representatives may change the positions they occupy remain the same. The position can be fixed by hashing the name of the cloud, which remains static. Now, if we route queries on this R-Ring, we can ensure that any data item always hashes onto the same cloud.

The process of creating an R-Ring is similar to that of the main DHT rings and the protocols are well understood. It is created whenever the system is initialized. However, note that this can lead to significant loads on the rendezvous nodes selected to take a part on the R-Ring. To prevent this, we balance load across various rendezvous nodes of the cloud by constructing an R-Ring for each rendezvous node of the cloud. All rendezvous nodes will occupy the positions specified by hash of the *cloudname* in each of the R-Rings they belong to, so that all of them yield the same mappings of keys to clouds. Then a query can be routed on any of these R-Rings thus balancing the load between different rendezvous nodes for each cloud.

2.2 Routing Protocols

In this section, we discuss the exact protocols with which a query is generated, routed and responded to. Also we discuss other issues like sizes of clouds and the system parameters used to control it.

2.2.1 Query: Crossover Peers

A query in the system originates in a cloud and is later brought out either on the DHT ring (services with discovery of service mechanisms) or the R-Rings (dynamic services) and it terminates in another cloud. An important issue in this regard is the selection of the peer which will perform the first crossover from the cloud to the ring. There are a number of important considerations - (1) the crossover should not be done by the querying peer, since it takes away its anonymity (2) it should be done in a way that load caused by querying is shared almost equally among the members of the cloud and (3) to prevent multiple copies of the query in the system, it should be done by a unique peer.

We enable this using a random walk in the cloud. Concretely, the querying peer makes the query message, sets up a random TTL[2] for the message and forwards it

[2]Number of application-level hops, as used in Gnutella

to one of its neighbors, selected randomly. The TTL is selected randomly to cloak the origin of the message. The neighbor decrements the TTL by 1 and again selects one of its neighbors randomly and forwards the message to it. Now after a few hops, the TTL will reduce to zero. The peer, at which that happens, is responsible for crossing over and taking the query to the ring. Such a peer is called a *crossover peer*. This mechanism ensures that there is only a unique peer performing the query. Secondly, because of the random walk the load will be distributed equally amongst the members of the cloud. Also similar to Gnutella, the peers in the query path cache its information, which is used to forward the reply back to the querying peer.

In case of services with discovery of service mechanisms, the query is tagged with a cloud name. Therefore, at the crossover, only a DHT lookup on the cloud name would suffice. In case of dynamic services, the query goes out on an R-Ring and the crossover peer might not have its routing table (only the rendezvous nodes have routing tables for R-Rings). Then, it can query a rendezvous node of its cloud for *only* the first step of the lookup and the crossover peer can continue the rest of the query in the typical DHT iterative fashion. We call this phase the *ring phase*. At the end of the ring phase, the crossover peer would have found the rendezvous node for the cloud of the service provider. The query is then forwarded to the rendezvous node, which will *broadcast* the query in its cloud. This broadcast, similar to Gnutella broadcasts, can be an expensive step effecting the scalability of the system. We tackle this issue in Section-2.2.2.

The crossover peer, while forwarding the query to the rendezvous node of the response cloud, includes its IP address and a port number where a reply can directly be sent. Clearly, this does not compromise any anonymity because the crossover peer is not the peer initiating the query. However, this saves us critical time and number of messages, since the reply can be directly sent to the crossover peer without any intermediate ring phase. Then, the reply can be forwarded back to the querying peer.

In the response cloud, every peer would get the query message because of the broadcast. The peer wishing to provide the desired service can then make the reply message (with IP address and port information for the querying crossover peer) and start a random walk with a random TTL, similar to the walk used while initiating a query. This way the service provider is also anonymized. It is important to note that the system provides guaranteed lookup of data i.e. for every data item available, a lookup will always succeed. This is because the query is first routed to the *appropriate* cloud using the discovery of service mechanisms or R-Rings and then *broadcasted* in that cloud, which ensures that every peer in the cloud receives the query and can respond.

2.2.2 Size of the Clouds

As mentioned before, since the query is broadcasted in the response cloud, we need to make sure that the size of the cloud does not become too large to make the broadcast costs prohibitive. In order to control the size of the cloud, we use two parameters:

- *R-Diameter*: It is the maximum distance of any peer from *any* rendezvous node. The distance is measured in number of application level hops in the underlying topology. It is denoted by r_{diam} and serves as the "length" dimension of the cloud. This parameter is important, since any query broadcasted by the rendezvous node will have an upper bound of r_{diam} on the number of hops required to reach any peer of the cloud.
- *Degree*: It is the maximum number of neighbors, a peer can have in the cloud. It is denoted by m and serves as the "breadth" dimension, controlling the density of the cloud.

Restricting r_{diam} and m to reasonable values will restrict the size of the cloud. In Section-3, we will show empirically how these parameters effect the overall costs and try to get best possible values.

To enforce these parameters, every peer keeps a vector of its distance from the rendezvous nodes and stops accepting new neighbors when the limits are reached. The distance vector is easy to compute in a recursive fashion. In every ping cycle[3], a peer computes its distance vector from the distance vectors of its neighbors. Now a peer which has m neighbors or is at r_{diam} distance from a rendezvous node will not accept any new incoming peers and the cloud is said to be *saturated*. Small temporary aberrations can be tolerated, since they will be short-lived and hence, not cause a major performance dip.

3 Performance Analysis

3.1 Scalability

We show the scalability of our system in both number of hops and number of messages transmitted analytically, in Appendix-A. For our empirical study, we created various Chord networks varying the number of nodes from a few to 5,000. We then added-on clouds to each topology. Each peer becomes a member of some cloud and a cloud is created when all existing ones are saturated. Then we ran a large number of queries in the system and computed the average costs. The queries were selected randomly, that is, a peer belonging to some cloud is randomly selected to query for a service by another random

[3]Gnutella requires each members to periodically ping its neighbors to check for node failures. As a result, no extra messages are used.

peer. Figure-2 shows the various costs for dynamic services (based on R-Rings). As it can be seen the costs for our System and Chord follow the same trend and differ by a constant. In this figure, we used an r_{diam} of 7 and m of 5. We have also depicted the number of hops on the R-Ring. This number is smaller than for Chord, since the number of clouds formed (size of R-Ring) was smaller than the total number of nodes in the system (size of Chord ring).

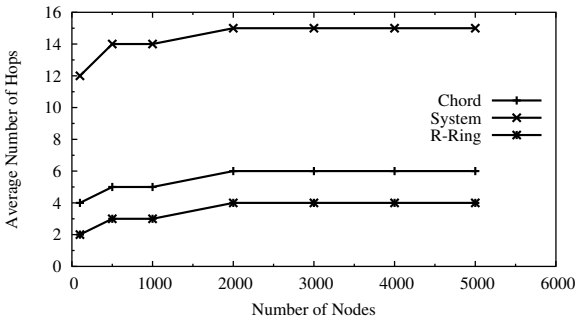

Figure 2: System Scalbility: Hops

The constants are due to the random walks in the query cloud and the response cloud and the tracing back of the reply. Since their lengths are bounded by r_{diam}, the average difference would always be close to $\frac{1}{2} * (3 * r_{diam}) \approx 10$ in this case. This shows that our system is as scalable as typical DHT systems and confirms our analysis.

Next we look at scalability in terms of aggregate number of messages transmitted for a single query transaction. This includes the messages exchanged during the random walks, tracing back of the reply and most critically the broadcast of the query. Figure-3 shows that our system is as scalable as DHT systems. The overall trend is similar to Chord, in which case number of messages is equal to the number of hops. Also, a big component of the costs is the number of messages in the Response Cloud, which includes the primary costs of broadcasting. The number of messages in the Query Cloud and on the R-Ring are small in comparison. Note that for bigger networks (P2P networks are millions in strength), the gap would be insignificant. For smaller networks, we can have different cloud topologies e.g. a hierarchy of DHT rings and unstructured clouds at the lowest level.

3.2 Effect of System Parameters

Figure-4 shows the average number of hops when r_{diam} is varied from 3 to 9 with m fixed at 5 for a 5,000 node topology. Increasing r_{diam} allows more peers to be added in a single cloud, increasing the cloud size. From the figure, we see an interesting trend. While the cost of Chord stays the same (r_{diam} does not effect the main DHT ring), our system begins to take more number

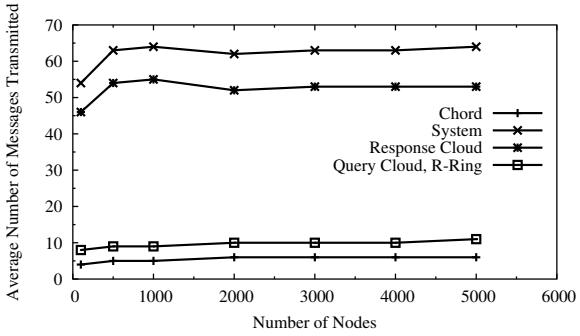

Figure 3: System Scalability: Messages

of hops even when the number of hops on the R-Ring decreases.

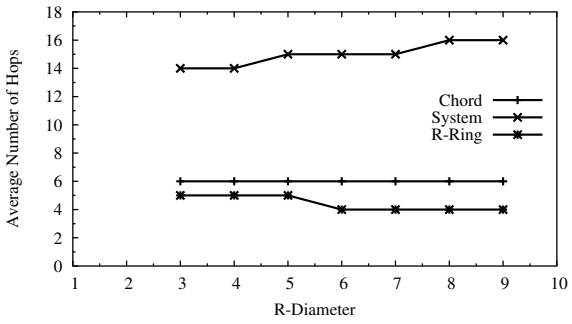

Figure 4: Effect of R-diameter on number of hops

This happens because with the increase in r_{diam}, even though the total number of clouds drops (because each cloud can accommodate more), the average lengths of the random walks and the number of hops before the broadcast reaches the responding peer in the response cloud also increases. This increase offsets the decrease in R-Ring lookup hops. While this would indicate that we should keep r_{diam} to a minimum, notice that it effects the level of anonymity offered by a cloud. Very small r_{diam} values lead to very small clouds and that provides little anonymity.

Next, we look at how r_{diam} effects the total number of messages transmitted. Figure-5 shows the average number of messages transmitted for a similar topology. As can be seen, with the increase in r_{diam}, the number of messages increases linearly, with the main component being the messages transmitted in the response cloud.

Next, we plot similar graphs for varying values of the degree m with r_{diam} fixed at 7 (5,000 nodes). Figure-6 depicts the effect of the degree on the average number of hops. Note that the increase in m decreases the average number of hops and infact there is a drastic decrease in the number of hops on the R-Ring. This occurs since increasing m allows clouds to become very dense and allows more and more peers to join the same cloud. This reduces the number of clouds very quickly.

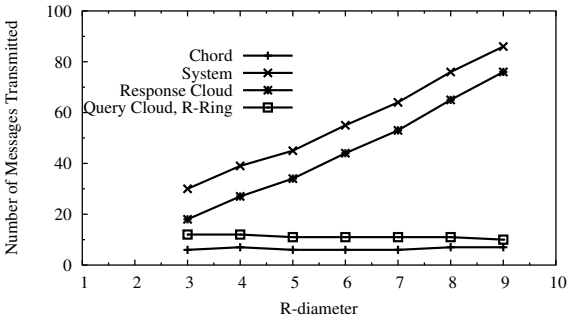

Figure 5: Effect of R-diameter on number of messages

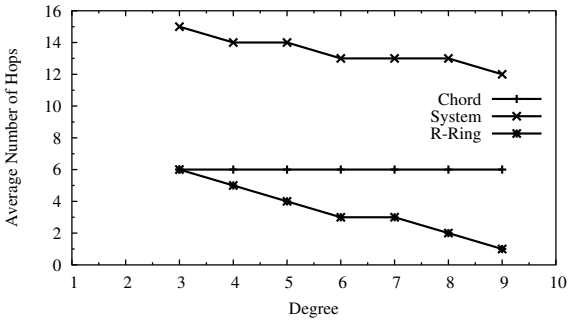

Figure 6: Effect of Degree on number of hops

However, with the clouds becoming more dense, there is a penalty to be paid in regards to the number of messages transmitted. This is because now, a large number of messages will be transmitted in a response cloud because of the broadcast of query messages. This can actually be seen in Figure-7, where increasing m drastically increases the total number of messages, with the biggest component being the broadcasting. In addition, larger m demands more resources from the peers, since they keep m open connections at all times.

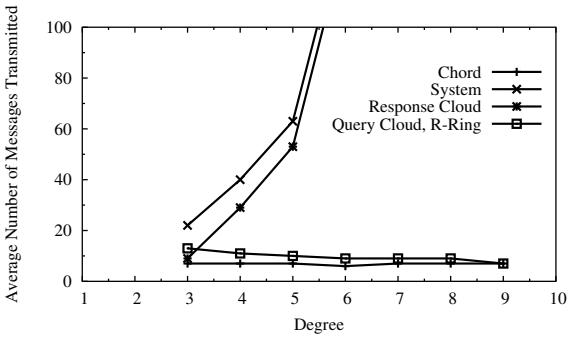

Figure 7: Effect of Degree on number of messages

This analysis indicates that we can control system performance by varying the two parameters with varying m providing fast changes and varying r_{diam} allowing for smaller fine tuning. Also, it appears that smaller r_{diam} and m values, as limited by the amount of anonymity required, would be the best.

4 Conclusions and Future Work

We have presented the design and development a decentralized sytem that provides mutually anonymous services over structured P2P networks and still ensures scalable lookup and guaranteed location of data. We first identified critical topology properties which are essential for mutual anonymity and introduce clouds to incorporate such properties into our systems. We described a number of mechanisms to enhance the scalability and efficiency of routing between and within clouds, ensuring the guaranteed lookup of data. We showed that our system is as scalable as DHT based systems in terms of both number of hops and aggregate messaging costs (differing only by constants) and also studied the effect of various system parameters. In future, we intend to work on security aspects of the system, aiming to provide mutually anonymous and secure services over structured decentralized overlays.

References

[1] M. Castro, P.Druschel, A. Kermarrec, and A. Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE JSAC*, 2002.

[2] I. Clarke, O. Sandberg, B. Wiley, and T.W. Hong. Freenet: A distributed anonymous information storage and retrieval system. *LNCS*, 2001.

[3] D. Karger et al. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *STOC*, 97.

[4] I. Stoica et al. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of SIGCOMM Annual Conference on Data Communication*, Aug 2001.

[5] N. Harvey et al. Skipnet: A scalable overlay network with practical locality properties. In *Proceedings of USITS*, 2003.

[6] S. Ratnasamy et al. Application-level multicast using content addressable networks. *LNCS*, 2001.

[7] B. Gedik and L. Liu. Peercq: A decentralized and self-configuring peer-to-peer information monitoring system. In *Proceedings of ICDCS*, 2003.

[8] Gnutella. http://gnutella.wego.com/, 2002.

[9] Kazaa. http://www.kazaa.com/, 2002.

[10] A. Keromytis, V. Misra, and D. Rubenstein. Sos: Secure overlay services. In *Proceedings of SIGCOMM*, 2002.

[11] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of SIGCOMM Annual Conference on Data Communication*, Aug 2001.

[12] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *International Conference on Distributed Systems Platforms*, 2001.

[13] SETI@Home. http://setiathome.ssl.berkeley.edu/, 1999.

[14] A. Singh, M. Srivatsa, L. Liu, and T. Miller. Apoidea: A Decentralized Peer-to-Peer Architecture for Crawling the World Wide Web. *Lecture Notes in Computer Science*, 2924, 2004.

A Analytical Analysis of Scalability

We analyze system scalability in terms of both the number of hops each query transaction requires and the aggregate number of messages used. A single query transaction includes the costs for both the query message and the reply message to reach their appropriate destinations. First, we look at the number of hops. We can divide the cost in the following components:

- *Query Cloud Hops:* It is the number of hops in the querying cloud. It is denoted by h_{query}. If the length of the initial random walk is q_{rand}, then clearly $h_{query} = 2 * q_{rand}$, since the reply is traced back on the same path.
- *Ring Lookup Hops:* It is the number of hops on the DHT ring or the R-Ring, once the crossover happens. It is denoted by h_{ring}.
- *Response Cloud Hops:* It is the number of hops in the response cloud. It is denoted by h_{resp} and is equal to the sum of hops due to the broadcast of the query in the response cloud (h_{bcast}) and the random walk initiated for the reply message (r_{rand}); that is, $h_{resp} = h_{bcast} + r_{rand}$.

Also remember that there can be two more hops required when (1) the crossover peer in the querying cloud queries the rendezvous node for the first hop for an R-Ring and (2) a peer in the response cloud sends the reply directly to the crossover peer. Hence, the total number of hops is given by:

$hops = h_{query} + h_{ring} + h_{resp} + 2$

$hops = 2 * q_{rand} + h_{ring} + h_{bcast} + r_{rand} + 2$

Now, we restrict the random walks by r_{diam}, since that would traverse the whole length of the cloud. Therefore, $1 \leq q_{rand} \leq r_{diam}$ and $1 \leq r_{rand} \leq r_{diam}$. Also, $0 \leq h_{bcast} \leq r_{diam}$, since r_{diam} is the farthest the responding peer can be. As a result,

$hops \leq 2 * r_{diam} + h_{ring} + r_{diam} + r_{diam} + 2$

$hops \leq 4 * r_{diam} + 2 + h_{ring}$

Also we know that $h_{ring} = O(logN)$, where N is the number of nodes on the ring. In case, the query occurs on the DHT ring, N is the total number of nodes in the system. In case of R-Rings, N is the number of clouds, since there is one node per cloud in the R-Ring. Also, typical r_{diam} values will be small constants like 7. As a result, $hops = O(logN)$, which implies that our system is as scalable as DHTs.

The analysis is interesting, specially for dynamic services. It shows that the total costs are $O(logN)$, where N is the number of clouds as opposed to the total number of nodes for normal DHTs. In case the number of clouds is less than the total number of nodes by a big margin, it can compensate for differing constants and potentially take lesser number of hops than normal DHT based systems! However, the caveat is that lesser number of clouds implies greater number of nodes in each cloud, which would require increasing r_{diam} and m values to accommodate them. This increases the differing constants and also increases the messaging costs.

For aggregate number of messages used, similar analysis will hold. In the query cloud and the ring phase there is one message per hop. The only difference is because of the broadcast of the query in the response cloud. Since size of the clouds[4] is bounded by constant parameters, our system will still be similarly scalable. We omit the exact proof because of space constraints. Note that the differing constant in this case would be much higher because of the broadcast in the response cloud.

[4] In a saturated cloud with Gnutella like topology, there can be at most $m^{r_{diam}}$ nodes